

# Certified Soundness of Simplest Known Formulation of First-Order Logic

**Larsen, John Bruntse**

*Published in:*
Proceedings of the ESSLLI 2017 Student Session

*Publication date:*
2017

*Document Version*
Publisher's PDF, also known as Version of record

# Certified Soundness of Simplest Known Formulation of First-Order Logic

John Bruntse Larsen

DTU Compute, Technical University of Denmark, 2800 Kongens Lyngby, Denmark

**Abstract.** In 1965, Donald Monk published a paper about an axiomatic system for first-order predicate logic that he described as "the simplest known formulation of ordinary logic". In this paper we show work in progress on certifying soundness of this system in the interactive proof assistant Isabelle. Through this work we demonstrate the usefulness of using proof assistants for validating mathematical results. This work also establishes an outline for future work such as a certified completeness proof of the axiomatic system in Isabelle.

**Keywords:** first-order logic, axiomatic system, soundness, proof assistant, Isabelle

## 1 Introduction

First-order predicate logic has a fundamental role in mathematics and computer science. It formalizes the concept of entities with properties and relations to other entities. It provides a framework for formal reasoning and proofs which are relevant in areas such as software engineering and AI. For this reason it is important that formalizations of first-order logic are correct so that they do not give erroneous results. By using an interactive proof assistant like Isabelle [2], we can work with formalizations in a certified manner as motivated by Geuvers [7] and Pfenning [6].

In this work we investigate using proof assistants for verifying a formulation of first-order logic with equality by J. Donald Monk [1]. In 1965, Monk published a paper about an axiomatic system for first-order predicate logic with equality that he described as "the simplest known formulation of ordinary logic" [3]. Simplicity in this context is to be understood as the simplicity of checking correctness of a proof. In the formulation, there are 10 axioms of which 2 of them have side conditions and the simplicity of the formulation is in particular the simplicity of checking these side conditions. To clarify the notion of simplicity in Monk's formulation, we compare it with a textbook formulation of natural deduction and a formulation of sequent calculus.

The goal of this paper is to show work in progress on using the interactive proof assistant Isabelle to verify soundness of Monk's formulation. The motivation for this work is to demonstrate the usefulness of working with a formulation from the 1960's in a modern proof assistant. We begin with introducing some

of notions from [3–5] that are crucial for understanding Monk's formulation and make comparisons with natural deduction and sequent calculus. We then describe how to verify soundness of the formulation in Isabelle where we use the following approach:

1. Defining the syntax of normalized first-order logic formulas.
2. Defining the semantics of the normalized formulas.
3. Defining the axioms and rules of Monk's axiomatic system and a proof system for it.
4. Defining and proving soundness of the proof system.

By proving soundness of the proof system based on Monk's axiomatic system, we obtain certified soundness of Monk's formulation. We found the following Isabelle commands useful in making the formalization:

**datatype** for defining a grammar in a BNF-style.
**abbreviation** for defining abbreviations of terms such as $\top \equiv \neg\bot$.
**fun** for defining recursive functions.
**definition** for defining non-recursive functions.
**inductive** for defining inductive constructs.
**lemma** for stating auxiliary lemmas for the soundness proof.
**theorem** for stating soundness.

In the following sections Isabelle commands are written in bold font. Finally we describe our work in progress on the proof of soundness in Isabelle and relate to other works on formalization.

## 2 Normalized Formulas and Axiomatic System

The formulation for the axiomatic system by Monk [3] uses notions and expressions from Tarski [5], and Kalish and Montague [4]. In the context of our work, the notion of *normalized* formulas is the most important notion to understand. It relies on a logical validity in first-order predicate logic with equality that transforms arbitrary predicates into a logically equivalent formula where the predicate variables can be replaced with the arity of the predicate. For example, any binary predicate can be transformed as follows.

$$P(x, y) \equiv \forall v_0(v_0 = x \rightarrow \forall v_1(v_1 = y \rightarrow P(v_0, v_1)))$$

In the formula on the right-hand side, the variables $v_0$ and $v_1$ can be considered implicit arguments of $P$ so that it is only necessary to represent $P$ by its name and its arity.

To illustrate consider the case of stating that $x + y = y + x$ where $x$ and $y$ are natural numbers. Let $\mathit{Plus}(x, y, z)$ be true iff $x + y = z$ where $x$, $y$ and $z$ are natural numbers. The statement can then be written as $\forall x, y, z(\mathit{Plus}(x, y, z) \leftrightarrow \mathit{Plus}(y, x, z))$. To get the normal form for Monk's formulation, we transform it

into an equivalent expression as outlined above. Note that the term lists of the *Plus* predicates in the new formula are syntactically identical.

$$\forall x, y, z(\mathit{Plus}(x, y, z) \leftrightarrow \mathit{Plus}(y, x, z)) \equiv$$
$$\forall x, y, z(\forall v_0(v_0 = x \rightarrow \forall v_1(v_1 = y \rightarrow \forall v_2(v_2 = z \rightarrow \mathit{Plus}(v_0, v_1, v_2)))) \leftrightarrow$$
$$\forall v_0(v_0 = y \rightarrow \forall v_1(v_1 = x \rightarrow \forall v_2(v_2 = z \rightarrow \mathit{Plus}(v_0, v_1, v_2)))))$$

Monk defines the axiomatic system $\Lambda_1$ shown in Table 1, where $A$, $B$, and $C$ are normalized formulas and $x$, $y$, and $z$ are variables. The occurrence check in (C5$^1$) for a predicate with arity $n$ takes $n$ comparisons due to the equalities in the transformation.

| | | |
|---|---|---|
| Axioms | | |
| (C1) | $(A \rightarrow B) \rightarrow (B \rightarrow C) \rightarrow A \rightarrow C$ | |
| (C2) | $A \rightarrow \neg A \rightarrow B$ | |
| (C3) | $(\neg A \rightarrow A) \rightarrow A$ | |
| (C4) | $\forall x(A \rightarrow B) \rightarrow \forall x A \rightarrow \forall x B$ | |
| (C5$^1$) | $A \rightarrow \forall x A$ | $x$ does not occur in $A$ |
| (C5$^2$) | $\neg\forall x A \rightarrow \forall x \neg\forall x A$ | |
| (C5$^3$) | $\forall x \forall y A \rightarrow \forall y \forall x A$ | |
| (C6) | $\neg\forall x \neg(x = y)$ | |
| (C7) | $x = y \rightarrow x = z \rightarrow y = z$ | |
| (C8) | $x = y \rightarrow A \rightarrow \forall x(x = y \rightarrow A)$ | $x$ is different from $y$ |
| Rules | | |
| (R1) | From $A$ and $A \rightarrow B$ infer $B$ | |
| (R2) | From $A$ infer $\forall x A$ | |

**Table 1.** Monk's axiomatic system $\Lambda_1$ for first-order predicate logic with equality. Note that (C5$^1$) only requires checking for occurrence, no matter if $x$ is bound or free.

## 3 On Simplicity Compared to Natural Deduction

In this section we compare Monk's formulation to natural deduction, as presented in a popular textbook on logic in computer science [11], in order to further clarify the notion of simplicity. To begin with, we highlight the use of substitution in natural deduction:

> Given a variable $x$, a term $t$ and a formula $\phi$ we define $\phi[t/x]$ to be the formula obtained by replacing each free occurrence of variable $x$ in $\phi$ with $t$.

On top of this definition there is a definition of what it means that '$t$ must be free for $x$ in $\phi$':

Given a term $t$, a variable $x$ and a formula $\phi$, we say that $t$ is free for $x$ in $\phi$ if no free $x$ leaf in $\phi$ occurs in the scope of $\forall y$ or $\exists y$ for any variable $y$ occurring in $t$.

Having a definition of substitution the natural deduction rules are defined as follows:

$$\frac{\begin{array}{|c|}\hline \neg\phi \\ \vdots \\ \bot \\ \hline\end{array}}{\phi}\ PBC \qquad \frac{\phi \qquad \phi \to \psi}{\psi}\ \to E \qquad \frac{\begin{array}{|c|}\hline \phi \\ \vdots \\ \psi \\ \hline\end{array}}{\phi \to \psi}\ \to I$$

$$\frac{\phi \vee \psi \qquad \begin{array}{|c|}\hline \phi \\ \vdots \\ \chi \\ \hline\end{array} \qquad \begin{array}{|c|}\hline \psi \\ \vdots \\ \chi \\ \hline\end{array}}{\chi}\ \vee E \qquad \frac{\phi}{\phi \vee \psi}\ \vee I_1 \qquad \frac{\psi}{\phi \vee \psi}\ \vee I_2$$

$$\frac{\phi \wedge \psi}{\phi}\ \wedge E_1 \qquad \frac{\phi \wedge \psi}{\psi}\ \wedge E_2 \qquad \frac{\phi \qquad \psi}{\phi \wedge \psi}\ \wedge I$$

$$\frac{\exists x\, \phi \qquad \begin{array}{|cc|}\hline x_0 & \phi\,[x_0/x] \\ & \vdots \\ & \chi \\ \hline\end{array}}{\chi}\ \exists E \qquad \frac{\phi\,[t/x]}{\exists x\, \phi}\ \exists I$$

$$\frac{\forall x\, \phi}{\phi\,[t/x]}\ \forall E \qquad \frac{\begin{array}{|cc|}\hline x_0 & \\ & \vdots \\ & \phi\,[x_0/x] \\ \hline\end{array}}{\forall x\, \phi}\ \forall I$$

Side conditions to rules for quantifiers:

$\exists E$: $x_0$ does not occur outside its box (and therefore not in $\chi$).
$\exists I$: $t$ must be free for $x$ in $\phi$.
$\forall E$: $t$ must be free for $x$ in $\phi$.
$\forall I$: $x_0$ is a new variable which does not occur outside its box.

In addition there is a special copy rule:

> A final rule is required in order to allow us to conclude a box with a formula which has already appeared earlier in the proof. [...] The copy rule entitles us to copy formulas that appeared before, unless they depend on temporary assumptions whose box has already been closed.

For truth, negation and biimplication, the following equivalences can be used where $A$ and $B$ are arbitrary formulas:

$$\begin{array}{rcl} \top & \equiv & \bot \to \bot \\ \neg A & \equiv & A \to \bot \\ A \leftrightarrow B & \equiv & (A \to B) \wedge (B \to A) \end{array}$$

There are a number of ways that the above formulation of natural deduction can be compared to Monk's formulation in terms of the simplicity of checking correctness of a proof:

**Checking for free occurrence of variables** The rules for the quantifiers rely on side conditions that rely on checking for free occurrence of variables in a formula. In Monk's formulation there are no side conditions that rely on this check. There is a check for occurrence of variables but without the condition that they must be free.

**Substitution** In addition to the above, the side condition for universal quantifier introduction also relies on substitution with a variable that does not occur outside its box. Axiom scheme C5[1] in Monk's formulation relies on checking for occurrence of a variable in a formula but it does not rely on substitution.

**Copy rule** The copy rule does not have any immediately corresponding rule in Monk's formulation. Checking correctness of the application of the copy rule involves checking if the formula in question has appeared earlier in the proof outside a box that has already been closed. There is some resemblance to applying a rule in Monk's formulation in that it also involves referring to some previously appearing formulas but there is no copying involved.

## 4 On Simplicity Compared to Sequent Calculus

Having looked at how Monk's formulation compares to natural deduction in terms of simplicity, we make a similar comparison with sequent calculus as presented by Tom Ridge [15].

Sequent calculus is a proof system in which the intention is to start from the goal formula to be proven and then apply rules that break down the proof into subgoals until there are no more goals left to be proven. Initially the goal formula is transformed into negation normal form, yielding a formula in which the only operators are $\exists$, $\forall$, $\wedge$, $\vee$, or $\neg$ and all negations are applied to predicates. Let the initial current sequent be a singleton list containing this formula. A proof is then a list of applications of the rules shown below to the current sequent, where $\Gamma$ and $\Delta$ are possibly empty lists. Applying any of the two rules in the top

(marked with a *) removes a sequent and the other six rules replace the current sequent with the sequent(s) in the top. The proof is completed when there are no more sequents left.

The intention behind the system is that you start with the rules for $\exists$, $\forall$, $\wedge$, and $\vee$ until the top four rules are applicable. The two rules $NoAx$ and $\overline{NoAx}$ are for "skipping" through formulas in the sequent either until one of the rules $Ax$ or $\overline{Ax}$ can remove the sequent or possibly never.

| Rule (* = high priority) | Comments |
|---|---|
| $\dfrac{}{\vdash P(v_{i_1},...,v_{i_k}), \Gamma, \overline{P}(v_{i_1},...,v_{i_k}), \Delta}\ Ax\ *$ | Leaf of the derivation tree. |
| $\dfrac{}{\vdash \overline{P}(v_{i_1},...,v_{i_k}), \Gamma, P(v_{i_1},...,v_{i_k}), \Delta}\ \overline{Ax}\ *$ | Leaf of the derivation tree. |
| $\dfrac{\vdash \Gamma, P(v_{i_1},...,v_{i_k})}{\vdash P(v_{i_1},...,v_{i_k}), \Gamma}\ NoAx$ | |
| $\dfrac{\vdash \Gamma, \overline{P}(v_{i_1},...,v_{i_k})}{\vdash \overline{P}(v_{i_1},...,v_{i_k}), \Gamma}\ \overline{NoAx}$ | |
| $\dfrac{\vdash \Gamma, A, B}{\vdash A \vee B, \Gamma}\ \vee$ | |
| $\dfrac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash A \wedge B, \Gamma}\ \wedge$ | The only branching rule. |
| $\dfrac{\vdash \Gamma, [v_i/x]A, (\exists x.A)^{i+1}}{\vdash (\exists x.A)^i, \Gamma}\ \exists$ | Superscripts are only relevant for this rule, and allow $[v_i/x]A$ to be instantiated for all $i$. |
| $\dfrac{\vdash \Gamma, [v_r/x]A}{\vdash \forall x.A, \Gamma}\ \forall$ | $v_r$ is a fresh free variable, chosen as $r = max(S)+1$, where $S$ is the set of subscripts already used for the free variables in $A$ ($r = 0$ if there are no free variables in $A$). |

We note the following when comparing the above notion of sequent calculus and Monk's axiomatic system in terms of simplicity of checking a proof:

**Checking for free occurrence of variables** Analogous to the comparison with natural deduction, the side condition for $\forall$ rely on checking for free occurrence of a variable in formula $A$ whereas Monk's axiomatic system only relies on checking for occurrence.

**Substitution** Similarly, the rules for $\exists$ and $\forall$ rely on substitution whereas Monk's axiomatic system does not.

**Scope** In sequent calculus all rules only refers to the current sequent. Specifically the two rules $NoAx$ and $\overline{NoAx}$ are used for changing the current sequent to the head of $\Gamma$. In comparison, the rules in Monk's axiomatic system refer to any of the previously appearing formulas.

## 5 Syntax and Semantics

In order to define the axiomatic system in Isabelle, we first define the syntax of normalized formulas. We define the syntax as a type using the Isabelle keyword **datatype** with two **abbreviation**s for truth and falsity. Thus values of this type represent normalized formulas. Note that the *nat* in the constructor for a predicate refers to the arity of the predicate, while the *nat*s in the constructors for equality and universal formulas refer to variables.

**datatype** *form* = *Pre string nat* | *Eq nat nat* | *Neg form* | *Imp form form* | *Uni nat form*

**abbreviation** (*input*) *Falsity* ≡ *Uni 0* (*Neg* (*Eq 0 0*))

**abbreviation** (*input*) *Truth* ≡ *Neg Falsity*

Although only the syntax is necessary in order to define the axiomatic system, we define the semantics next in order to show the meaning of the normalized formulas. The semantics is defined as a recursive function: it takes an environment, an interpretation, and a formula as input and calculates a truth value. The cases for equality and complex formulas are straightforward but the case for predicates follows from the normalized form. In this form, the number denotes the arity $n$ of the predicate and thus implies that the variables for the predicate are $v_0, ..., v_{n-1}$. Thus we can evaluate a predicate by mapping $v_0, ..., v_{n-1}$ to terms with the environment and then map the predicate name and terms to a truth value with the interpretation. An auxiliary recursive function for calculating the list of variables is used in this mapping.

**fun** *vars* :: *nat* ⇒ *nat list*
**where**
*vars 0* = [] |
*vars n* = (*vars* (*n*−*1*)) @ [*n*−*1*]

**fun** *semantics* :: (*nat* ⇒ *'a*) ⇒ (*string* ⇒ *'a list* ⇒ *bool*) ⇒ *form* ⇒ *bool*
**where**
*semantics e g* (*Eq x y*) ⟷ (*e x*) = (*e y*) |
*semantics e g* (*Pre p arity*) ⟷ *g p* (*map e* (*vars arity*)) |
*semantics e g* (*Neg f*) ⟷ ¬ *semantics e g f* |
*semantics e g* (*Imp p q*) ⟷ (*semantics e g p* ⟶ *semantics e g q*) |
*semantics e g* (*Uni x p*) ⟷ (∀ *t*. (*semantics* (*e*(*x*:=*t*)) *g p*))

## 6 Axioms and Rules

Having definitions for syntax and semantics of normalized formulas, we define the axiomatic system and proof system.

To begin with, we define a recursive function for the occurrence check. The function should return true only if the given variable does not occur in the given formula. Recall that predicate formulas are defined only in terms of arity as described earlier. Thus in the occurrence check for predicate *Pre p n* with variable $x$, it suffices to check if $x < n$. In order to save space the body of the function has been omitted.

**fun** *not-occurs-in* :: *nat* ⇒ *form* ⇒ *bool*

Next we define a type that represents theorems. The idea is that the axiomatic system produces values of this type from its axiom schemes and rules, and thus the axiomatic system produces theorems. Inapplicable use of the rules and axiom schemes with side conditions will result in trivial truth.

**datatype** *thm* = *Thm*(*concl*: *form*)

**abbreviation** (*input*) *fail-thm* ≡ *Thm Truth*

The two rules of the axiomatic system are defined as functions. Given the input theorems they produce a new theorem. We define the rules using the Isabelle keyword **definition**.

**definition** *modusponens* :: *thm* ⇒ *thm* ⇒ *thm*
**where**
*modusponens s s′* ≡ *case concl s of Imp p q* ⇒
*let p′* = *concl s′ in if p* = *p′ then Thm q else fail-thm* | - ⇒ *fail-thm*

**definition** *gen* :: *nat* ⇒ *thm* ⇒ *thm*
**where**
*gen x a* ≡ *Thm* (*Uni x* (*concl a*))

We define the ten axioms schemes in a similar manner. These are functions that produce a theorem given a number of formulas. For the axiom schemes without side conditions it suffices that the formulas are well-founded (being of the type *form*). For example for C1 and C2 we have

**definition** *c1* :: *form* ⇒ *form* ⇒ *form* ⇒ *thm*
**where**
*c1 p q r* ≡ *Thm* (*Imp* (*Imp p q*) (*Imp* (*Imp q r*) (*Imp p r*)))

**definition** *c2* :: *form* ⇒ *thm*
**where**
*c2 p* ≡ *Thm* (*Imp* (*Imp* (*Neg p*) *p*) *p*)

For the axiom schemes with a side condition we additionally require that the formulas satisfy the side condition. This is done by using if-statements with

*fail-thm* as the failure value. For (C5$^2$) we perform the occurrence check with *not-occurs-in* described earlier, and for (C8) we use non-equality. For example for C8 we have

**definition** *c8* :: *nat* ⇒ *nat* ⇒ *form* ⇒ *thm*
**where**
*c8 x y p* ≡ *if x* ≠ *y then Thm* (*Imp* (*Eq x y*) (*Imp p* (*Uni x* (*Imp* (*Eq x y*) *p*)))) *else fail-thm*

Thus far we have defined the axioms schemes and rules of the axiomatic system. We tie them all together in a proof system that given a formula returns true if the axiomatic system can produce a theorem with it and false otherwise. In the following section we define soundness of the axiomatic system in terms of this proof system. In Isabelle, we define the proof system inductively with the **inductive**-keyword. The annotation (⊢ - *0*) at the end states that ⊢ can be used as notation for *OK* with precedence 0 (lowest). The inductive definition states the formulas that follow from the axiomatic system. The first two cases state that the formulas of theorems produced by the rules follow from the axiomatic system. The remaining ten cases state that the formulas produced by the axiom schemes follow from the axiomatic system.

**inductive** *OK* :: *form* ⇒ *bool* (⊢ - *0*)
**where**
*case-modusponens*:
⊢ *concl f* ⟹ ⊢ *concl f'* ⟹ ⊢ *concl* (*modusponens f f'*) |
*case-gen*:
⊢ *concl f* ⟹ ⊢ *concl* (*gen* - *f*) |
*case-c1*:
⊢ *concl* (*c1* - - -) |
*case-c2*:
⊢ *concl* (*c2* -) |
*case-c3*:
⊢ *concl* (*c3* - -) |
*case-c4*:
⊢ *concl* (*c4* - - -) |
*case-c5-1*:
⊢ *concl* (*c5-1* - -) |
*case-c5-2*:
⊢ *concl* (*c5-2* - -) |
*case-c5-3*:
⊢ *concl* (*c5-3* - - -) |
*case-c6*:
⊢ *concl* (*c6* - -) |
*case-c7*:
⊢ *concl* (*c7* - - -) |
*case-c8*:
⊢ *concl* (*c8* - - -)

Thus we have defined an axiomatic system for first-order predicate logic with normalized formulas in Isabelle based on Monk's formulation in [3], and a proof

system that we can proceed to prove soundness of and thus certify soundness of the axiomatic system.

## 7 Proving Soundness

Having defined the axiomatic system and the proof system for it in Isabelle, we can certify soundness by using interactive proof assistance in Isabelle. Our approach to proving soundness is by providing auxiliary lemmas about the definitions so that Isabelle automatically can check that soundness follows from those lemmas and the definitions. Isabelle assists by keeping track of the proof as it is written and what subgoals remain to be proven. These subgoals can be used as templates for the auxiliary lemmas which in some cases can be proven just by applying Isabelle library lemmas. For example the following lemma is solved by applying the Isabelle library function *fun-upd-idem*. The hypothesis $x \neq y$ is not needed in the lemma. It is rather a product of using the subgoal from Isabelle directly as a template for the lemma.

**lemma** *update-identity*: $x \neq y \longrightarrow e\ x = e\ y \longrightarrow semantics\ e\ g\ p \longrightarrow semantics\ (e(x := e\ y))\ g\ p$
**by** (*simp add*: *fun-upd-idem*)

Finally we define soundness of the proof system as follows.

**theorem** *soundness*: $\vdash p \implies semantics\ e\ g\ p$

The proof itself is a work in progress.

## 8 Related Work

In this section we consider other works in the literature that investigate formalizing proof systems for first-order logic in proof assistants, and how they differ from the work in this paper. The work in this paper follow in line with the work of Villadsen, Jensen and Schlichtkrull [9], who used Isabelle to formalize and generate code for the kernel of the LCF-style prover for first-order logic with equality by Harrison [8]. Their work is now also part of the Archive of Formal Proofs [10]. The work in this paper differs from their work in that we have investigated a formalization of Monk's original formulation from 1965 rather than Harrison's which is based on Monk and Tarski. Other work on computer assisted formalizations based on Monk and Tarski include the work of Normal Megill [14] on using the Metamath language for archiving, verifying, and studying mathematical proofs.

Earlier we compared Monk's formulation with natural deduction and sequent calculus which have already been formalized in Isabelle. Villadsen, Jensen and Schlichtkrull [18] used Isabelle to prove soundness of a proof system for natural deduction. They developed the browser-based software NaDeA where one could make natural deduction proofs that would then be checked by the proof system

verified by Isabelle. For sequent calculus, Tom Ridge and James Margetson [16] made a formalization of sequent calculus in Isabelle for which they proved soundness and completeness that is now also part of the Archive of Formal Proofs [15]. The formalization was later used as a starting point by Villadsen, Schlichtkrull and From [19] for a prover made with code-generation with the aim of being used for teaching logic and verification to bachelor computer science students.

Looking at work with Isabelle formalizations of other proof systems we have Blanchette et. al. [12] who investigated how codatatypes can be used for proving soundness and completeness of different kinds of proof systems for first-order logic in Isabelle. In comparison, we have focused on proving soundness of Monk's formulation, and we have not investigated the use of codatatypes in proving soundness, although it is certainly worth considering in future work. There is also the work of Schlichtkrull [17] about a formalization of the resolution calculus for first-order logic that includes soundness and completeness. Looking even further at verification of proof systems in other frameworks we have the work of Kumar, Arthan, Myreen and Owens [13] on formalization of higher-order logic in HOL4.

## 9 Conclusion

The work in progress described in this paper shows the usefulness of working with axiomatic systems in a proof assistant. We have defined syntax and semantics of the first-order logic formulas and the axiomatic system. The proof assistant checks for type-correctness of the definitions and provides assistance in making a soundness proof of the axiomatic system. The soundness proof itself is a work in progress. In this way, we can certify soundness of Monk's axiomatic system in Isabelle.

There is room for improvement in the current formalization. Currently we use "truth as failure" when a rule is not applicable as a mechanism to ensure soundness. Using a dependently-typed logical framework could potentially make this trick redundant and also simplify some of the rules. It is also worth considering how the formalization could include other terms than just variables. Going further, the work can be extended to show additional useful properties of the axiomatic system, such as completeness.

## Acknowledgements

This work is part of the Industrial PhD project *Hospital Planning with Multi-Agent Goals* between PDC A/S and Technical University of Denmark. We are grateful to Innovation Fund Denmark for funding and the governmental institute Region H, which manages the hospitals in the Danish capital region, for being a collaborator on the project. We would like to thank PDC A/S for providing feedback on the ideas described in this paper. We would also like to thank Jørgen Villadsen and Anders Schlichtkrull for comments on a draft. Finally we would like to thank the referees for in-depth comments on the paper that we think helped improving it substantially.

## References


1. Monk, J.: Mathematical Logic. Graduate Texts in Mathematics, Springer (1976)
2. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
3. Monk, J.D.: Substitutionless predicate logic with identity. Archiv Für Mathematische Logik Und Grundlagenforschung 7(3-4), 102–121 (1965)
4. Kalish, D., Montague, R.: On Tarski's formalization of predicate logic with identity. Archiv Für Mathematische Logik Und Grundlagenforschung 7(3-4), 81–101 (1965)
5. Tarski, A.: A simplified formalization of predicate logic with identity. Archiv Für Mathematische Logik Und Grundlagenforschung 7(1-2), 61–79 (1965)
6. Pfenning, F.: Chapter 17 - logical frameworks. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 1063 – 1147. North-Holland (2001)
7. Geuvers, H.: Proof assistants: History, ideas and future. Sadhana 34(1), 3–25 (2009)
8. Harrison, J.: Handbook of Practical Logic and Automated Reasoning. Cambridge University Press (2009)
9. Jensen, A.B., Schlichtkrull, A., Villadsen, J.: Verification of an LCF-style first-order prover with equality. Isabelle Workshop (2016)
10. Jensen, A.B., Schlichtkrull, A., Villadsen, J.: First-order logic according to Harrison. Archive of Formal Proofs (Jan 2017), `http://isa-afp.org/entries/FOL_Harrison.shtml`, Formal proof development
11. Huth, M., Ryan, M.: Logic in Computer Science : Modelling and Reasoning About Systems. Second Edition. Cambridge University Press, (2004)
12. Blanchette, J.C., Popescu, A., Traytel, D.: Soundness and completeness proofs by coinductive methods. Journal of Automated Reasoning 58(1), 149–179 (2017)
13. Kumar, R., Arthan, R., Myreen, M.O., Owens, S.: Self-formalisation of higher-order logic: Semantics, soundness, and a verified implementation. Journal of Automated Reasoning 56(3), 221–259 (2016)
14. Megill, N.D.: Metamath: A Computer Language for Pure Mathematics. Lulu Press, Morrisville, North Carolina (2007), `http://us.metamath.org/downloads/metamath.pdf`
15. Ridge, T.: A mechanically verified, efficient, sound and complete theorem prover for first order logic. Archive of Formal Proofs (Sep 2004), `http://isa-afp.org/entries/Verified-Prover.shtml`, Formal proof development
16. Ridge, T., Margetson, J.: A mechanically verified, sound and complete theorem prover for first order logic. In: Hurd, J., Melham, T. (eds.) TPHOL's 2005. LNCS, vol. 3603, pp. 294–309. Springer (2005)
17. Schlichtkrull, A.: Formalization of the resolution calculus for first-order logic. In: International Conference on Interactive Theorem Proving. LNCS, vol. 9807, pp. 341–357. Springer (2016)
18. Villadsen, J., Jensen, A.B., Schlichtkrull, A.: NaDeA: A natural deduction assistant with a formalization in Isabelle. IfCoLog Journal of Logics and their Applications 4(1), 55–82 (2017)
19. Villadsen, J., Schlichtkrull, A., From, A.H.: Code generation for a simple first-order prover. Isabelle Workshop (2016)